RPC BROKER

GETTING STARTED WITH THE BROKER DEVELOPMENT KIT (BDK)

Version 1.1; Patch XWB*1.1*40

September 1997

Revised February 2005

Department of Veterans Affairs
VistA Health Systems Design & Development (HSD&D)
Infrastructure and Security Services (ISS)

Revision History

Documentation Revisions

The following table displays the revision history for this document. Revisions to the documentation are based on patches and new versions released to the field.

Date	Revision	Description	Author(s)
09/97	1.0	Initial RPC Broker Version 1.1 software release.	Thom Blom and Joel Ivey, Oakland OIFO
05/01/02	2.0	Revised Version for RPC Broker Patch XWB*1.1*13.	Thom Blom and Joel Ivey, Oakland OIFO
05/08/02	3.0	Revised Version for RPC Broker Patch XWB*1.1*26.	Thom Blom and Joel Ivey, Oakland OIFO
02/24/05	4.0	Revised Version for RPC Broker Patches XWB*1.1*35 and 40. Also, reviewed document and edited for the "Data Scrubbing" and the "PDF 508 Compliance" projects. **Data Scrubbing—**Changed all patient/user TEST data to conform to HSD&D standards and conventions as indicated below: • The first three digits (prefix) of any Social Security Numbers (SSN) start with "000" or "666." • Patient or user names are formatted as follows: NHEPATIENT,[N] or NHEUSER,[N] respectively, where the N is a number written out and incremented with each new entry (e.g., NHEPATIENT, ONE, NHEPATIENT, TWO, etc.). • Other personal demographic-related data (e.g., addresses, phones, IP addresses, etc.) were also changed to be generic. **PDF 508 Compliance—**The final PDF document was recreated and now supports the minimum requirements to be 508 compliant (i.e., accessibility tags, language selection, alternate text for all images/icons, fully functional Web links, successfully passed Adobe Acrobat	Thom Blom and Joel Ivey, Oakland OIFO

Date	Revision	Description	Author(s)
		Quick Check).	

Table i: Documentation revision history

Patch Revisions

For the current patch history related to this software, please refer to the Patch Module on FORUM.

Contents

Revision History iii
Figures and Table ix
Acknowledgements xi
Orientation xiii

1. Introduction 1-1

About this Version of the BDK 1-1
Features 1-2
Backward Compatibility Issues 1-2

2. RPC Broker Components for Delphi 2-3

TRPCBroker Component 2-3
TRPCBroker Properties and Methods 2-3
TRPCBroker Key Properties 2-4
TRPCBroker Key Methods 2-4
How to Connect to an M Server 2-5
TCCOWRPCBroker Component 2-6
Single Signon/User Context (SSO/UC) 2-7
TSharedBroker Component 2-7
TSharedRPCBroker Component 2-7
TXWBRichEdit Component 2-8

3. Remote Procedure Calls (RPCs) 3-1

What is a Remote Procedure Call? 3-1
Create Your Own RPCs 3-1
Writing M Entry Points for RPCs 3-2
RPC Entry in the REMOTE PROCEDURE File 3-5
What Makes a Good Remote Procedure Call? 3-5
How to Execute an RPC from a Client Application 3-5
RPC Security: How to Register an RPC 3-6

4. Other RPC Broker APIs 4-1

GetServerInfo Function 4-1

VistA Splash Screen Procedures 4-2
XWB GET VARIABLE VALUE RPC 4-3
M Emulation Functions 4-3
Encryption Functions 4-4
$$BROKER^XWBLIB 4-4
$$RTRNFMT^XWBLIB 4-5

5. Debugging and Troubleshooting 5-1

How to Debug Your Client Application 5-1
Troubleshooting Connections 5-1

6. RPC Broker Developer Utilities 6-1

Programmer Settings 6-1

7. RPC Broker and Delphi 7-1

Delphi 6.0 Packages 7-1
Delphi V. 6 Standard Edition *Not* Recommended for BDK Development 7-1
XWB_Rxx.BPL File 7-1
Delphi 5.0 Packages 7-1
Delphi V. 5 Standard Edition *Not* Recommended for BDK Development 7-1
XWB_Rxx.BPL File 7-2
Delphi 4.0 Packages 7-2
XWB_Rxx.BPL File 7-2
Delphi 3.0 Packages 7-2
VistaBroker.DPL 7-2
Distributing the Delphi VCL30.DPL 7-3

8. RPC Broker Dynamic Link Library (DLL) 8-1

DLL Interface 8-1
Exported Functions 8-1
Header Files Provided 8-1
Sample DLL Application 8-1
Return Values from RPCs 8-2
COTS Development and the DLL 8-2

Glossary Glossary-1
Index Index-1

Figures and Table

Table i: Documentation revision history....iv
Table ii: Documentation symbol descriptions....xiii
Table iii: Commonly used RPC Broker terms....xiv
Figure 1-1: Delphi's Tool Properties dialogue....xvii
Table 2-1: TRPCBroker component key properties....2-4
Figure 2-1: OnCreate event handler—Sample code....2-6
Table 3-1: RPC Broker return value types....3-3
Table 3-2: Input parameter types....3-4
Figure 3-1: RPC M entry point example: Sum of two numbers....3-4
Figure 3-2: RPC M entry point example: Sorted array....3-4
Table 3-3: REMOTE PROCEDURE file key field entries....3-5
Figure 3-3: Param property: Sample settings....3-6
Figure 3-4: Exception handler—try...except code: Sample usage....3-6
Figure 3-5: BrokerExample application....3-8
Figure 4-1: Server and port configuration selection dialogue....4-1
Figure 4-2: VistA Splash screen....4-2
Figure 4-3: Displaying a VistA splash screen: Sample code....4-3
Figure 4-4: XWB GET VARIABLE VALUE RPC usage: Sample code....4-3
Figure 4-5: Encryption in VistA M Server: Sample code....4-4
Figure 4-6: Decryption in VistA M Server: Sample code....4-4
Figure 6-1: RPC Broker Programmer Preferences dialogue....6-1
Table 6-1: Programmer preference settings....6-1
Table 8-1: TRPCBroker component's Results property....8-2

Acknowledgements

The RPC Broker Development Team consists of the following Development and Infrastructure Service (DaIS) and Infrastructure & Security Services (ISS) personnel (listed alphabetically):

- ISS Program and Project Manager—Larry Weldon
- Centralized Planner Support Team (CPST)—Laura Rowland
- Developers—Alan Chan, Wally Fort, Joel Ivey (lead), and Raul Mendoza
- Functional Analysts—Lauren Hardeen and Lauren Gorgoglione
- Software Quality Assurance (SQA)—Matt Alderman
- Technical Writer—Thom Blom

The RPC Broker Development Team would like to thank the following sites/organizations/personnel for their assistance in reviewing and/or testing RPC Broker V. 1.1, Patch XWB*1.1*40 software and documentation (listed alphabetically):

- Computerized Patient Record System (CPRS) GUI—Development Team

Orientation

How to Use this Manual

Throughout this manual, advice and instructions are offered regarding the use of the RPC Broker V. 1.1 and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VistA).

There are no special legal requirements involved in the use of the RPC Broker.

This manual uses several methods to highlight different aspects of the material:

- Various symbols are used throughout the documentation to alert the reader to special information. The following table gives a description of each of these symbols:

Symbol	Description
	Used to inform the reader of general information including references to additional reading material
	Used to caution the reader to take special notice of critical information

Table ii: Documentation symbol descriptions

- Descriptive text is presented in a proportional font (as represented by this font).
- Conventions for displaying TEST data in this document are as follows:
  - The first three digits (prefix) of any Social Security Numbers (SSN) will begin with either "000" or "666."
  - Patient and user names will be formatted as follows: [Application Name]PATIENT,[N] and [Application Name]USER,[N] respectively, where "Application Name" is defined in the Approved Application Abbreviations document and "N" represents the first name as a number spelled out and incremented with each new entry. For example, in Kernel (KRN) test patient and user names would be documented as follows: KRNPATIENT,ONE; KRNPATIENT,TWO; KRNPATIENT,THREE; etc.
- Sample HL7 messages, "snapshots" of computer online displays (i.e., roll-and-scroll screen captures/dialogues) and computer source code, if any, are shown in a *non*-proportional font and enclosed within a box.

  Also included are Graphical User Interface (GUI) Microsoft Windows images (i.e., dialogues or forms).
  - User's responses to online prompts will be boldface.
  - The "**<Enter>**" found within these snapshots indicate that the user should press the Enter key on their keyboard. Other special keys are represented within < > angle brackets. For example, pressing the PF1 key can be represented as pressing **<PF1>**.

➢ Author's comments, if any, are displayed in italics or as "callout" boxes.

Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field and file names, and security keys (e.g., the XUPROGMODE key).
- Object Pascal code uses a combination of upper- and lowercase characters. All Object Pascal reserved words are in boldface type.

Commonly Used Terms

The following is a list of terms and their descriptions that you may find helpful while reading the RPC Broker documentation:

Term	Description
Client	A single term used interchangeably to refer to a user, the workstation (i.e., PC), and the portion of the program that runs on the workstation.
Component	A software object that contains data and code. A component may or may not be visible. For a more detailed description, please refer to the *Borland Delphi for Windows User Guide*.
GUI	The Graphical User Interface application that is developed for the client workstation.
Host	The term Host is used interchangeably with the term Server.
Server	The computer where the data and the RPC Broker remote procedure calls (RPCs) reside.

Table iii: Commonly used RPC Broker terms

Please refer to the "Glossary" for additional terms and definitions.

How to Obtain Technical Information Online

Exported file, routine, and global documentation can be generated through the use of Kernel, MailMan, and VA FileMan utilities.

Methods of obtaining specific technical information online will be indicated where applicable under the appropriate topic.

Please refer to the *RPC Broker Technical Manual* for further information.

Help at Prompts

VistA software provides online help and commonly used system default prompts. Users are encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of VistA software.

To retrieve online documentation in the form of Help in any VistA character-based product:

- Enter a single question mark ("**?**") at a field/prompt to obtain a brief description. If a field is a pointer, entering one question mark ("**?**") displays the HELP PROMPT field contents and a list of choices, if the list is short. If the list is long, the user will be asked if the entire list should be displayed. A **YES** response will invoke the display. The display can be given a starting point by prefacing the starting point with an up-arrow ("^") as a response. For example, **^M** would start an alphabetic listing at the letter M instead of the letter A while **^127** would start any listing at the 127th entry.
- Enter two question marks ("**??**") at a field/prompt for a more detailed description. Also, if a field is a pointer, entering two question marks displays the HELP PROMPT field contents and the list of choices.
- Enter three question marks ("**???**") at a field/prompt to invoke any additional Help text stored in Help Frames.

Obtaining Data Dictionary Listings

Technical information about files and the fields in files is stored in data dictionaries. You can use the List File Attributes option on the Data Dictionary Utilities submenu in VA FileMan to print formatted data dictionaries.

For details about obtaining data dictionaries and about the formats available, please refer to the "List File Attributes" chapter in the "File Management" section of the *VA FileMan Advanced User Manual*.

Assumptions About the Reader

This manual is written with the assumption that the reader is familiar with the following:

- VistA computing environment:
  - Kernel
  - VA FileMan data structures and terminology
- Microsoft Windows environment
- M programming language
- Object Pascal programming language.
- Borland Delphi Integrated Development Environment (IDE)

It provides an overall explanation of configuring RPC Broker and the functionality contained in RPC Broker Version 1.1. However, no attempt is made to explain how the overall VistA programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA home pages on the World Wide Web for a general orientation to VistA. For example, go to the Health Systems Design & Development (HSD&D) Home Page at the following Web address:

http://vista.med.va.gov/

Reference Materials

Readers who wish to learn more about the RPC Broker should consult the following:

- *RPC Broker Release Notes*
- *RPC Broker Installation Guide*
- *RPC Broker Systems Manual*
- *RPC Broker Technical Manual*
- *RPC Broker Getting Started with the Broker Development Kit (BDK)* (this manual)

- *RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help designed for programmers, distributed in the BDK). This manual provides an overview of development with the RPC Broker.

  You may want to make an entry for BROKER.HLP in Delphi's Tools Menu, to make it easily accessible from within Delphi. To do this, use Delphi's Tools | Configure Tools option. Create a new menu entry similar to the following:

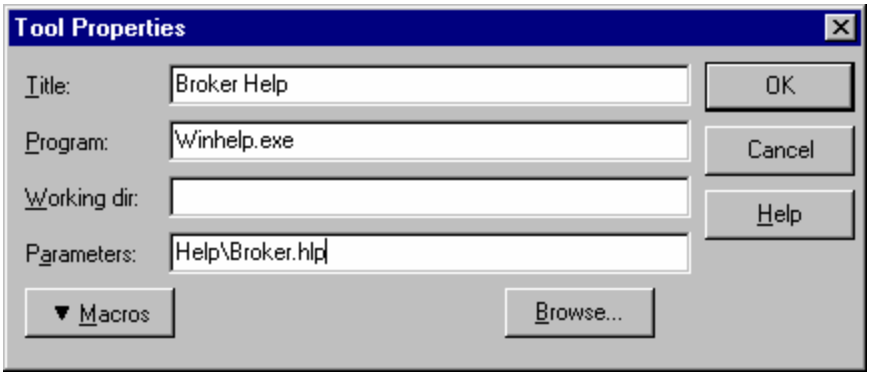

Figure 1-1: Delphi's Tool Properties dialogue

- BROKER.HLP as Context-sensitive Help within Delphi. The BROKER.HLP file provides context-sensitive help within Delphi on the TRPCBroker component and its associated properties and methods. This help is available when you have installed the RPC Broker V 1.1 BDK. When installed, you can select the TRPCBroker component or one of its properties in the Object Inspector, and press the F1 key to get help on that item.
- RPC Broker Home Page at the following Web address:

  http://vista.med.va.gov/broker/index.asp

  This site provides announcements, additional information (e.g., Frequently Asked Questions [FAQs], advisories), documentation links, archives of older documentation and software downloads.

VistA documentation is made available online in Microsoft Word format and in Adobe Acrobat Portable Document Format (PDF). The PDF documents *must* be read using the Adobe Acrobat Reader (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems Incorporated at the following web address:

http://www.adobe.com/

For more information on the use of the Adobe Acrobat Reader, please refer to the *Adobe Acrobat Quick Guide* at the following web address:

http://vista.med.va.gov/iss/acrobat/index.asp

VistA documentation can be downloaded from the Health Systems Design and Development (HSD&D) VistA Documentation Library (VDL) Web site:

http://www.va.gov/vdl/

VistA documentation and software can also be downloaded from the Enterprise VistA Support (EVS) anonymous directories:

- Albany OIFO ftp.fo-albany.med.va.gov
- Hines OIFO ftp.fo-hines.med.va.gov
- Salt Lake City OIFO ftp.fo-slc.med.va.gov
- Preferred Method download.vista.med.va.gov

  This method transmits the files from the first available FTP server.

DISCLAIMER: The appearance of external hyperlink references in this manual does *not* constitute endorsement by the Department of Veterans Affairs (VA) of this Web site or the information, products, or services contained therein. The VA does *not* exercise any editorial control over the information you may find at these locations. Such links are provided and are consistent with the stated purpose of this VA Intranet Service.

1. Introduction

The Remote Procedure Call (RPC) Broker (also referred to as "Broker") is a client/server system within VA's Veterans Health Information Systems and Technology Architecture (VistA) environment. It establishes a common and consistent foundation for client/server applications being written as part of VistA. It enables client applications to communicate and exchange data with M Servers.

This manual introduces developers to the RPC Broker and the Broker Development Kit (BDK). The emphasis is on using the RPC Broker in conjunction with Borland's Delphi software. However, the RPC Broker supports other development environments.

This manual provides an overview of development with the RPC Broker.

For more complete information on development with the RPC Broker components, please refer to the *RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help distributed with the BDK).

This document is intended for the VistA development community and Information Resource Management (IRM) staff. A wider audience of technical personnel engaged in operating and maintaining the Department of Veterans Affairs (VA) software may also find it useful as a reference.

About this Version of the BDK

Version 1.1 of the RPC Broker (fully patched) provides programmers with the capability to develop new VistA client/server software using the following RPC Broker Delphi components in the 32-bit environment (listed alphabetically):

- TCCOWRPCBroker
- TRPCBroker (original component)
- TSharedBroker
- TSharedRPCBroker
- TXWBRichEdit

These RPC Broker components wrap the functionality of the Broker resulting in a more modularized and orderly interface. Those components derived from the original TRPCBroker component, inherit the TRPCBroker properties and methods.

Features

This enhanced Broker software has the following functionality/features:

- Supports Single Sign-On/User context (SSO/UC)—As of Patch XWB*1.1*40, the TCCOWRPCBroker component enabled Single Sign-On/User Context (SSO/UC) in CCOW-enabled applications.
- Supports Non-Callback Connections—As of Patch XWB*1.1*35, the RPC Broker components are built with a UCX or non-callback Broker connection, so that it can be used from behind firewalls, routers, etc. This functionality is controlled via the new TRPCBroker component IsBackwardCompatibleConnection property.
- Supports Silent Logon capabilities—As of Patch XWB*1.1*13, the RPC Broker provides "Silent Login" capability. It provides functionality associated with the ability to make logins to a VistA M Server without the RPC Broker asking for Access and Verify code information.
- Documented Deferred RPCs and Capability to Run RPCs on a Remote Server.
- Multi-instances of the RPC Broker—As of Patch XWB*1.1*13, the RPC Broker code was modified to permit an application to open two separate Broker instances with the same Server/ListenerPort combination, resulting in two separate partitions on the server. Previously, an attempt to open a second Broker instance ended up using the same partition. For this capability to be useful for concurrent processing, an application would have to use threads to handle the separate Broker sessions.

  Although we believe there should be no problems, the RPC Broker is not yet guaranteed to be thread safe.

- Updated components, properties, methods, and types.
- Separate Design-time and Run-time Packages—As of Patch XWB*1.1*14, the BDK contains separate run-time and design-time packages.
- Supports Delphi V. 6.0 and 7.0

To develop VistA applications in a 32-bit environment you must have Delphi V. 2.0 or greater. This version of the RPC Broker component will *not* allow you to develop applications in Delphi V. 1.0. However, the Broker routines on the M server will continue to support VistA applications previously developed in the 16-bit environment.

The default installation of the Broker creates a separate BDK directory (i.e., BDK32) that contains the required Broker files for development.

Backward Compatibility Issues

Client applications compiled with this version of the RPC Broker (V. 1.1) will *not* work at a site that has not upgraded its RPC Broker server software to V.1.1.

On the other hand, client applications compiled with RPC Broker V.1.0 will work with the V. 1.1 RPC Broker server.

2. RPC Broker Components for Delphi

::

For more detailed information on the RPC Broker components for Delphi, please refer to the *RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help distributed with the BDK).

TRPCBroker Component

The main tool to develop client applications for the RPC Broker environment is the TRPCBroker component for Delphi. The TRPCBroker component adds the following abilities to your Delphi application:

- **Connecting to an M server**
  - Authenticate the user
  - Set up the environment on the server
  - Bring back the introductory text
- **Invoking Remote Procedure Calls (RPCs) on the M Server**
  - Send data to the M Server
  - Perform actions on the server
  - Return data from the server to the client

To add the TRPCBroker component to your Delphi application, simply drop it from the Kernel tab of Delphi's component palette to a form in your application.

TRPCBroker Properties and Methods

As a Delphi component, the TRPCBroker component is controlled and accessed through its properties and methods. By setting its properties and executing its methods, you can connect to an M server from your application and execute RPCs on the M server to exchange data and perform actions on the M server.

For most applications, you will only need to use a single TRPCBroker component to manage communications with the M server.

TRPCBroker Key Properties

The following table lists the most important properties of the TRPCBroker component.

For a complete list of all of Broker properties, please refer to the *RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help distributed with the BDK).

Property	Description
ClearParameters	If True, the Param property is cleared *after* every invocation of the Call, strCall, or the lstCall methods.
ClearResults	If True, the Results property is cleared *before* every invocation of the Call method, thus assuring that only the results of the last call are returned.
Connected	Setting this property to True connects your application to the server.
ListenerPort	Sets server port to connect to a Broker Listener process (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)
Param	Run-time array in which you set any parameters to pass as input parameters when calling an RPC on the server.
RemoteProcedure	Name of a RemoteProcedure entry that the Call, lstCall, or strCall method should invoke.
Results	This is where any results are stored after a Call, lstCall, or strCall method completes.
Server	Name of the server to connect to (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)

Table 2-1: TRPCBroker component key properties

TRPCBroker Key Methods

This section lists the most important methods of the TRPCBroker component.

For a complete list of all of Broker methods, please refer to the *RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help distributed with the BDK).

procedure Call;

This method executes an RPC on the server and returns the results in the TRPCBroker component's Results property.

Call expects the name of the remote procedure and its parameters to be set up in the RemoteProcedure and Param properties respectively. If ClearResults is True, then the Results property is cleared before the call. If ClearParameters is True, then the Param property is cleared after the call finishes.

function strCall: string;s:

This method is a variation of the Call method. Only use it when the return type is a single string. Instead of returning results in the TRPCBroker component's Results[0] property node, results are returned as the value of the function call. Unlike the Call method, the Results property is not affected; no matter the setting of ClearResults, the value is left unchanged.

procedure lstCall(OutputBuffer: TStrings);s:

This method is a variation of the Call method. Instead of returning results in the TRPCBroker component's Results property, it instead returns results in the TStrings object you specify. Unlike the Call method, the Results property is not affected; no matter the setting of ClearResults, the value is left unchanged.

function CreateContext(strContext: string): boolean;
s:

This method creates a context for your application. Pass an option name in the strContext parameter. If the function returns True, a context was created, and your application can use all RPCs entered in the option's RPC multiple.

Examples

For examples of how to use these methods to invoke RPCs, please refer to the "How to Execute an RPC from a Client Application" topic in the "Remote Procedure Calls (RPCs)" chapter of this manual.

How to Connect to an M Server

To establish a connection from your application to a Broker server:

1. From the Kernel component palette tab, add a TRPCBroker component to your form.
2. Add code to your application to connect to the server; one likely location is your form's OnCreate event handler. The code should:
   a. Use the GetServerInfo function to retrieve the run-time server and port to connect to. This function is not a method of the TRPCBroker component; it is described in the Other RPC Broker APIs chapter.
   b. Inside of an exception handler **try...except** block, set RPCBroker1's Connected property to True. This causes an attempt to connect to the Broker server.
   c. Check if an EBrokerError exception is raised. If this happens, connection failed. You should inform the user of this and then terminate the application.

The code, placed in an OnCreate event handler, should look like:

```
procedure TForm1.FormCreate(Sender: TObject);
var   ServerStr: String;
        PortStr: String;
begin
  // get the correct port and server from registry
  if GetServerInfo(ServerStr,PortStr)<>mrCancel then
  begin
    RPCBroker1.Server:=ServerStr;
    RPCBroker1.ListenerPort:=StrToInt(PortStr);
  end
  else Application.Terminate;

  // establish a connection to the Broker
  try
    RPCBroker1.Connected:=True;
  except
    On EBrokerError do
    begin
      ShowMessage('Connection to server could not be established!');
      Application.Terminate;
    end;
  end;
end;
```

Figure 2-1: OnCreate event handler—Sample code

3. A connection with the Broker M Server is now established. You can use the CreateContext method of the TRPCBroker component to authorize use of RPCs for your user, and then use the Call, lstCall, and strCall methods of the TRPCBroker component to execute RPCs on the M server. See the next chapter, Remote Procedure Calls, for information on creating and executing RPCs.

TCCOWRPCBroker Component

As of Patch XWB*1.1*40, the TCCOWRPCBroker component was added to Version 1.1 of the RPC Broker. The TCCOWRPCBroker Delphi component allows VistA application developers to make their applications CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware with all of the client/server-related functionality in one integrated component. Using the TCCOWRPCBroker component, an application can share User Context stored in the CCOW Context Vault.

Thus, when a VistA CCOW-enabled application is recompiled with the TCCOWRPCBroker component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).

This RPC Broker component is derived from the original TRPCBroker Component; it inherits the TRPCBroker properties and methods.

Single Signon/User Context (SSO/UC)

The Veterans Health Administration (VHA) information systems user community expressed a need for a single sign-on (SSO) service with interfaces to VistA, Health*e*Vet VistA, and non-VistA systems. This new architecture will allow users to authenticate and sign on to multiple applications that are CCOW-enabled and SSO/UC-aware using a single set of credentials, which reduces the need for multiple ID's and passwords in the Health*e*Vet clinician desktop environment. The RPC Broker software addressed this architectural need by providing a new TCCOWRPCBroker component in RPC Broker Patch XWB*1.1*40.

The TCCOWRPCBroker component allows VistA application developers to make their applications CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware with all of the client/server-related functionality in one integrated component. Using the TCCOWRPCBroker component, an application can share User Context stored in the CCOW Context Vault.

Thus, when a VistA CCOW-enabled application is recompiled with the TCCOWRPCBroker component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).

For more information on SSO/UC and making your Broker-based applications CCOW-enabled and SSO/UC-aware, please consult the *Single Sign-On/User Context (SSO/UC) Installation Guide* and *Single Sign-On/User Context (SSO/UC) Deployment Guide* on the VistA Documentation Library (VDL).

TSharedBroker Component

As of Patch XWB*1.1*26, the TSharedBroker component was added to Version 1.1 of the RPC Broker. The TSharedBroker Delphi component provides applications or plugins to applications easy access to an RPCBroker without the need for a separate M partition. Each component has its own security (i.e., option) as well. The default value of the AllowShared property is True. If an application will have RPCs that require extensive time, it would be best to *not* share a Broker instance and the AllowShared property should then be set to False.

This RPC Broker component is derived from the original TRPCBroker Component; it inherits the TRPCBroker properties and methods.

TSharedRPCBroker Component

As of Patch XWB*1.1*26, the TSharedBroker component was added to Version 1.1 of the RPC Broker. The TSharedRPCBroker Delphi component provides applications or plugins to applications easy access to an RPCBroker without the need for a separate M partition. Each component has its own security (i.e., option) as well. The default value of the AllowShared is True. If an application will have RPCs that require extensive time, it would be best to *not* share a Broker instance and the AllowShared property should then be set to False.

This RPC Broker component is derived from the original TRPCBroker Component; it inherits the TRPCBroker properties and methods.

TXWBRichEdit Component

As of Patch XWB*1.1*13, the TXWBRichEdit component was added to Version 1.1 of the RPC Broker. The TXWBRichEdit Delphi component replaces the Introductory Text Memo component on the Login Form. TXWBRichEdit is a version of the TRichEdit component that uses Version 2 of Microsoft's RichEdit Control and adds the ability to detect and respond to a Uniform Resource Locator (URL) in the text. This component permits us to provide some requested functionality on the login form. As an XWB namespaced component we are required to put it on the Kernel tab of the component palette, however, it rightly belongs on the Win32 tab.

3. Remote Procedure Calls (RPCs)

What is a Remote Procedure Call?

A remote procedure call (RPC) is a defined call to M code that runs on an M server. A client application, through the RPC Broker, can make a call to the M server and execute an RPC on the M server. This is the mechanism through which a client application can:

- Send data to an M server.
- Execute code on an M server.
- Retrieve data from an M server.

An RPC can take optional parameters to do some task and then return either a single value or an array to the client application. RPCs are stored in the REMOTE PROCEDURE file (#8994).

Relationship Between an M Entry Point and an RPC

An RPC can be thought of as a wrapper placed around an M entry point for use with client applications. Each RPC invokes a single M entry point. The RPC passes data in specific ways to its corresponding M entry point and expects any return values from the M entry point to be returned in a pre-determined format. This allows client applications to connect to the RPC Broker, invoke an RPC, and through the RPC, invoke an M entry point on a server.

Create Your Own RPCs

You can create your own custom RPCs to perform actions on the M server and to retrieve data from the M server. Then you can call these RPCs from your client application. Creating an RPC requires you to perform the following two steps:

1. Write and test the M entry point that is called by the RPC.
2. Add the RPC entry that invokes your M entry point, in the REMOTE PROCEDURE file (#8994).

Writing M Entry Points for RPCs

First Input Parameter for RPCs (Required)

The RPC Broker always passes a variable by reference in the first input parameter to your M routine. It expects results (one of five types described in Table 3-1) to be returned in this parameter. You must always set some return value into that first parameter before your routine returns.

Return Value Types for RPCs

There are five RETURN VALUE TYPES for RPCs as shown in the table below. Choose a return value type that is appropriate to the type of data your RPC needs to return to your client. Your M entry point should set the return value (in the routine's first input parameter) accordingly.

RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC WORD WRAP ON Setting	Value(s) returned in Client Results

Single Value	Set the return parameter to a single value.
	For example:
	`TAG(RESULT) ;`
	`  S RESULT="DOE, JOHN"`
	`  Q`
		No effect	Value of parameter, in Results[0].

Array	Set an array of strings into the return parameter, each subscripted one level descendant.
	For example:
	`TAG(RESULT) ;`
	`  S RESULT(1)="ONE"`
	`  S RESULT(2)="TWO"`
	`  Q`
	For large arrays consider using the GLOBAL ARRAY return value type to avoid memory allocation errors.
		No effect	Array values, each in a Results item.

Word-processing	Set the return parameter the same as you set it for the ARRAY type. The only difference is that the WORD WRAP ON field (#.08) setting affects the Word-processing return value type.	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].

<table>
<tr><th>RPC Return Value Type</th><th>How M Entry Point Should Set the Return Parameter</th><th>RPC WORD WRAP ON Setting</th><th>Value(s) returned in Client Results</th></tr>
<tr><td rowspan="2">Global Array</td><td rowspan="2">Set the return parameter to a closed global reference in ^TMP. The global's data nodes will be traversed using $QUERY, and all data values on global nodes descendant from the global reference are returned.

This type is especially useful for returning data from VA FileMan word processing fields, where each line is on a 0-subscripted node.

 The global reference you pass is killed by the Broker at the end of RPC Execution as part of RPC cleanup. Do not pass a global reference that is not in ^TMP or that should not be killed.

This type is useful for returning large amounts of data to the client, where using the ARRAY type can exceed the symbol table limit and crash your RPC.

For example, to return signon introductory text you could do:

<pre>TAG(RESULT);
 M ^TMP("A6A",$J)=
^XTV(8989.3,1,"INTRO")
 ;this node not needed
 K ^TMP("A6A",$J,0)
 S RESULT=$NA(^TMP("A6A",$J))
 Q</pre></td><td>True</td><td>Array values, each in a Results item.</td></tr>
<tr><td>False</td><td>Array values, concatenated into Results[0].</td></tr>
<tr><td>Global Instance</td><td>Set the return parameter to a closed global reference.

For example, to return the 0th node from the NEW PERSON file (#200) for the current user:

<pre>TAG(RESULT) ;
 S RESULT=$NA(^VA(200,DUZ,0))
 Q</pre></td><td>No effect</td><td>Value of global node, in Results[0].</td></tr>
</table>

Table 3-1: RPC Broker return value types

Input Parameter Types for RPCs (Optional)

The M entry point for an RPC can optionally have input parameters (i.e., beyond the first parameter, which is always used to return an output value). The client passes data to your M entry point through these parameters.

The client can send data to an RPC (and therefore your entry point) in one of the following three format types:

Param PType	Param Value
Literal	Delphi string value, passed as a string literal to the M server.
Reference	Delphi string value, treated on the M Server as an M variable name and resolved from the symbol table at the time the RPC executes.
List	A single-dimensional array of strings in the Mult subproperty of the Param property, passed to the M Server where it is placed in an array. String subscripting can be used.

Table 3-2: Input parameter types

The type of the input parameters passed in the Param property: of the TRPCBroker component determines the format of the data you must be prepared to receive in your M entry point.

RPC M Entry Point Examples

The following two examples illustrate sample M code that could be used in simple RPCs.

The following takes two numbers and returns their sum:

```
SUM(RESULT,A,B)    ;add two numbers
 S RESULT=A+B
 Q
```

Figure 3-1: RPC M entry point example: Sum of two numbers

The following example receives an array of numbers and returns them as a sorted array to the client:

```
SORT(RESULT,UNSORTED)    ;sort numbers
 N I
 S I=""
 F  S I=$O(UNSORTED(I)) Q:I=""  S RESULT(UNSORTED(I))=UNSORTED(I)
 Q
```

Figure 3-2: RPC M entry point example: Sorted array

RPC Entry in the REMOTE PROCEDURE File

After the M code is complete, you need to create the RPC itself in the REMOTE PROCEDURE file (#8994). The following fields in the REMOTE PROCEDURE file (#8994) are key to the correct operation of an RPC:

Field Name	Required?	Description
NAME (#.01)	Yes	The name that identifies the RPC (this entry should be namespaced in the package namespace).
TAG (#.02)	Yes	The tag at which the remote procedure call begins.
ROUTINE (#.03))	Yes	The name of the routine that should be invoked to start the RPC.
WORD WRAP ON (#.08)	No	Affects Global Array and Word-processing return value types only. If set to False, data is returned in a single concatenated string in Results[0]. If set to True, each array node on the M side is returned as a distinct array item in Results.
RETURN VALUE TYPE (#.04)	Yes	This indicates to the Broker how to format the return values. For example, if the RETURN VALUE TYPE is set as Word-processing, then each entry in the returning list will have a <CR><LF> (<carriage return><line feed>) appended.

Table 3-3: REMOTE PROCEDURE file key field entries

What Makes a Good Remote Procedure Call?

- Silent calls (no I/O to terminal or screen, no user intervention required).
- Minimal resources required (passes data in brief, controlled increments).
- Discrete calls (requiring as little information as possible from the process environment).
- Generic as possible (different parts of the same package as well as other packages could use the same RPC).

How to Execute an RPC from a Client Application

1. If your RPC has any input parameters beyond the mandatory first parameter, set a Param node in the TRPCBroker's Param property for each. For each input parameter, set the following sub properties:
    - Value
    - PType (Literal, List, or Reference).

If the parameter's PType is List, however, set a list of values in the Mult subproperty rather than setting the Value subproperty.

Here is an example of some settings of the Param property::

```
RPCBroker1.Param[0].Value := '10/31/97';
RPCBroker1.Param[0].PType := literal;
RPCBroker1.Param[1].Mult['"NAME"'] := 'SMITH, JOHN';
RPCBroker1.Param[1].Mult['"SSN"'] := '123-45-6789';
RPCBroker1.Param[1].PType := list;
```

Figure 3-3: Param property: Sample settings

2. Set the TRPCBroker's RemoteProcedure property to the name of the RPC to execute.

   ```
   RPCBroker1.RemoteProcedure:='A6A LIST';
   ```

3. Invoke the Call method of the TRPCBroker component to execute the RPC. All calls to the Call method should be done within an exception handler **try...except** statement, so that all communication errors (which trigger the EBrokerError exception) can be trapped and handled. For example:

```
try
    RPCBroker1.Call;
except
  On EBrokerError do
    ShowMessage('A problem was encountered communicating with the server.');
end;
```

Figure 3-4: Exception handler—try...except code: Sample usage

4. Any results returned by your RPC are returned in the TRPCBroker component's Results property. Depending on how you set up your RPC, results are returned either in a single node of the Results property (Result[0]) or in multiple nodes of the Results property.

   You can also use the lstCall and strCall methods to execute an RPC. The main difference between these methods and the Call method is that lstCall and strCall do not use the Results property, instead returning results into a location you specify.

RPC Security: How to Register an RPC

Security for RPCs is handled through the RPC registration process. Each client application must create a context for itself, which checks if the application user has access to a "B"-type option in the Kernel menu system. Only RPCs assigned to that option can be run by the client application.

To enable your application to create a context for itself:

1. Create a "B"-type option in the OPTION file (#19) for your application.

   The OPTION TYPE "**B**" represents a **B**roker client/server type option.

2. In the RPC multiple for this option type, add an entry for each RPC that your application calls. You can also specify a security key that can lock each RPC (this is a pointer to the SECURITY KEY file [#19.1]) and M code in the RULES subfield that can also determine whether to enable access to each RPC.

3. When you export your software using KIDS, export both your RPCs and your software option.

4. Your application must create a context for itself on the server, which checks access to RPCs. In the initial code of your client application, make a call to the CreateContext methods: of your TRPCBroker component. Pass your application's "B"-type option's name as a parameter. For example:

   ```
   RPCBroker1.CreateContext(option_name)
   ```

   If the CreateContext methods: returns True, only those RPCs designated in the RPC multiple of your application option will be permitted to run.

   If the CreateContext methods: returns False, you should terminate your application (if you don't your application will run, but you will get errors every time you try to access an RPC).

5. End-users of your application must have the "B"-type option assigned to them on one of their menus, in order for the CreateContext methods: to return True.

Bypassing RPC Security for Development

Having the XUPROGMODE security key allows you to bypass the Broker security checks. You can run any RPC without regard to application context (without having to use the CreateContext method). This is a convenience for application development. When you complete development, make sure you test your application from an account *without* the XUPROGMODE keys:, to ensure that all RPCs needed are properly registered.

BrokerExample Online Code Example

The BrokerExample sample application (i.e., BROKEREXAMPLE.EXE) provided with the BDK demonstrates the basic features of developing RPC Broker-based applications, including:

- Connecting to an M server.
- Creating an application context.
- Using the GetServerInfo function.
- Displaying the VistA splash screen.

- Setting the TRPCBroker Param property for each Param PType (literal, reference, list).
- Calling RPCs with the Call method.
- Calling RPCs with the lstCall and strCall methods.

The client source code files for the BrokerExample application are located in the SAMPLES\RPCBROKER\BROKEREX subdirectory of the main BDK32 directory.

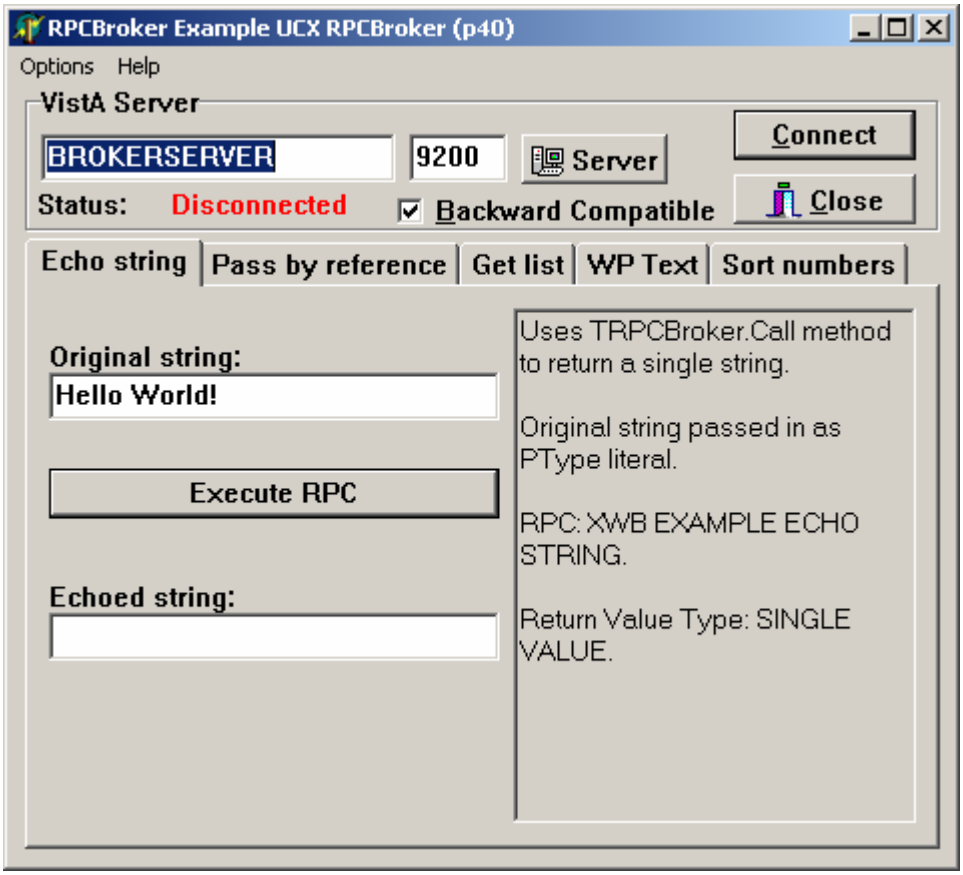

Figure 3-5: BrokerExample application

4. Other RPC Broker APIs

GetServerInfo Function

The GetServerInfo function retrieves the end-user workstation's server and port. Use this function to set the TRPCBroker component's Server and ListenerPort properties to reflect the end-user workstation's settings before connecting to the server.

If there is more than one server/port to choose from, GetServerInfo displays dialogue that allows users to select a service to connect to, as shown below:

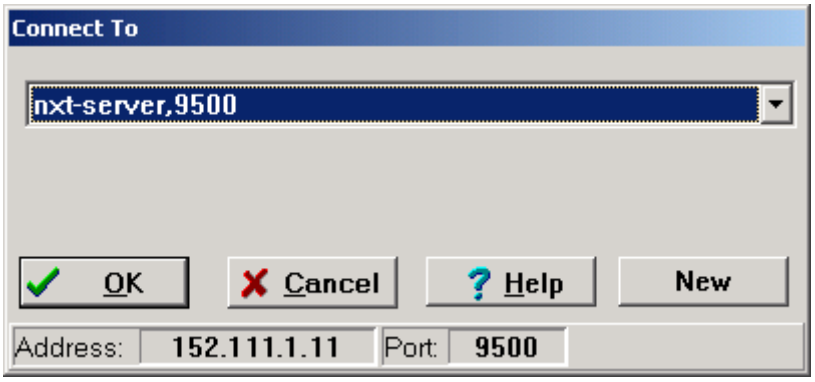

Figure 4-1: Server and port configuration selection dialogue

If exactly one server and port entry is defined in the Microsoft Windows Registry, GetServerInfo does *not* display this dialogue. The values in the single Microsoft Windows Registry entry are returned, with no user interaction required.

If more than one server and port entry exists in the Microsoft Windows Registry, the dialogue is displayed, and the user chooses to which server they want to connect.

If no values for server and port are defined in the Microsoft Windows Registry, GetServerInfo does not display this dialogue, and automatic default values are returned (i.e., BROKERSERVER and 9200).

Syntax of GetServerInfo function:

```
function GetServerInfo(var Server, Port: string): integer;
```

The unit is RpcConf1.

VistA Splash Screen Procedures

Two procedures in SplVista.PAS unit are provided to display a VistA splash screen when an application loads:

```
procedure SplashOpen;
procedure SplashClose(TimeOut: longint);
```

It is recommended that the splash screen be opened and closed in the section of Pascal code in an application's project file (i.e., .DPR).

To use the splash screen in an application:

1. Open your application's project (.DPR) file (in Delphi, choose View | Project Source).

2. Include the SplVista in the uses clause of the project source.

3. Call SplashOpen immediately after the first form of your application is created and call SplashClose just prior to invoking the Application.Run methods:.

4. Use the TimeOut parameters: to ensure a minimum display time.

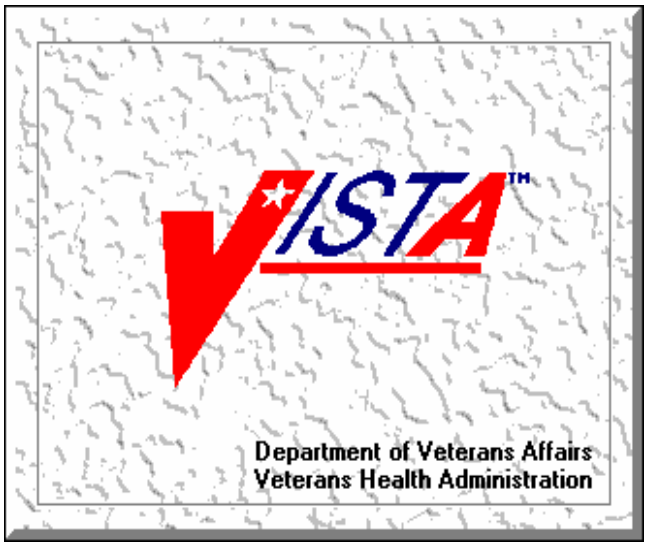

Figure 4-2: VistA Splash screen

```
uses
  Forms, Unit1 in 'Unit1.pas', SplVista;

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  SplashOpen;
  SplashClose(2000);
  Application.Run;
end.
```

Figure 4-3: Displaying a VistA splash screen: Sample code

XWB GET VARIABLE VALUE RPC

You can call the XWB GET VARIABLE VALUE RPC (distributed with the RPC Broker) to retrieve the value of any M variable in the server environment. Pass the variable name in Param[0].Value and the type (reference) in Param[0].PType. Also, the current context of your user must give them permission to execute the XWB GET VARIABLE VALUE RPC (it must be included in the RPC multiple of the "B"-type option registered with the CreateContext function).

For example:

```
RPCBroker1.RemoteProcedure := 'XWB GET VARIABLE VALUE';
RPCBroker1.Param[0].Value :='DUZ';
RPCBroker1.Param[0].PType := reference;
try
    RPCBroker1.Call;
except
  On EBrokerError do
    ShowMessage('Connection to server could not be established!');
end;
ShowMessage('DUZ is '+RPCBroker1.Results[0]);
```

Figure 4-4: XWB GET VARIABLE VALUE RPC usage: Sample code

M Emulation Functions

Piece Function

The Piece function is a scaled down Pascal version of M's $PIECE function. It is declared in MFUNSTR.PAS.

```
function Piece(x: string; del: string; piece: integer) : string;
```

Translate Function

The Translate function is a scaled down Pascal version of M's $TRANSLATE function. It is declared in MFUNSTR.PAS.

```
function Translate(passedString, identifier, associator: string): string;
```

Encryption Functions

Kernel and the RPC Broker provide some rudimentary encryption and decryption functions. Data can be encrypted on the client end and decrypted on the server, and vice-versa.

In Delphi

Include HASH in the "uses" clause of the unit in which you'll be encrypting or decrypting.

Function prototypes are as follows:

```
function Decrypt(EncryptedText: string): string;

function Encrypt(NormalText: string): string;
```

On the VistA M Server

To encrypt:

```
>S CIPHER=$$ENCRYP^XUSRB1("Hello world!") W CIPHER

/U'llTG~TVl&f-
```

Figure 4-5: Encryption in VistA M Server: Sample code

To decrypt:

```
>S PLAIN=$$DECRYP^XUSRB1(CIPHER) W PLAIN

Hello world!
```

Figure 4-6: Decryption in VistA M Server: Sample code

$$BROKER^XWBLIB

Use this function in the M code called by an RPC to determine if the Broker is executing the current process. It returns 1 if this is true, 0 if false.

$$RTRNFMT^XWBLIB

Use this function in the M code called by an RPC to change the return value type that the RPC will return on-the-fly. This allows you to change the return value type to any valid return value type (Single Value, Array, Word-processing, Global Array, or Global Instance). It also lets you set WORD WRAP ON to True or False, on-the-fly, for the RPC.

For more information about $$RTRNFMT^XWBLIB, please refer to the *RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help distributed with the BDK).

5. Debugging and Troubleshooting

How to Debug Your Client Application

Beside the normal debugging facilities provided by Delphi, you can also invoke a debug mode so that you can step through your code on the client side and your RPC code on the M server side simultaneously.

To do this:

1. On the client side, set the DebugMode property on the TRPCBroker component to True. When the TRPCBroker component connects with this property set to True, you will get a dialogue indicating your workstation IP address and the port number.
2. At this point, switch over to the M server and set any break points in the routines being called in order to help isolate the problem. Then issue the M debug command (e.g., ZDEBUG in DSM).
3. Start the following M server process:

   ```
   >D EN^XWBTCP
   ```

   You will be prompted for the workstation IP address and the port number. After entering the information, switch over to the client application and click on the OK button.
4. You can now step through the code on your client and simultaneously step through the code on the server side for any RPCs that your client calls.

RPC Error Trapping

M errors on the VistA M Server that occur during RPC execution are trapped by the use of M and Kernel error handling. In addition, the M error message is sent back to the Delphi client. Delphi will raise an exception EBrokerError and a popup box displaying the error. At this point RPC execution terminates and the channel is closed.

Troubleshooting Connections

Identifying the Listener Process on the Server

On DSM systems, where the Broker Listener is running, the Listener process name is RPCB_Port:NNNN, where NNNN is the port number being listened to. This should help quickly locate Listener processes when troubleshooting any connection problems.

Identifying the Handler Process on the Server

On DSM systems the name of a Handler process is ipXXX.XXX:NNNN, where XXX.XXX are the last two octets of the client IP address and NNNN is the port number.

Testing Your RPC Broker Connection

To test the RPC Broker connection from your workstation to the M Server, use the RPC Broker Diagnostic Program (RPCTEST.EXE).

For a complete description of the RPC Broker Diagnostic program, please refer to the "Troubleshooting" chapter in the *RPC Broker Systems Manual*.

6. RPC Broker Developer Utilities

Programmer Settings

You can use BrokerProgPref.EXE to define certain default property values for the TRPCBroker component. When you place TRPCBroker component(s) on your form(s) in Delphi, the settings you define are used as the default property values.

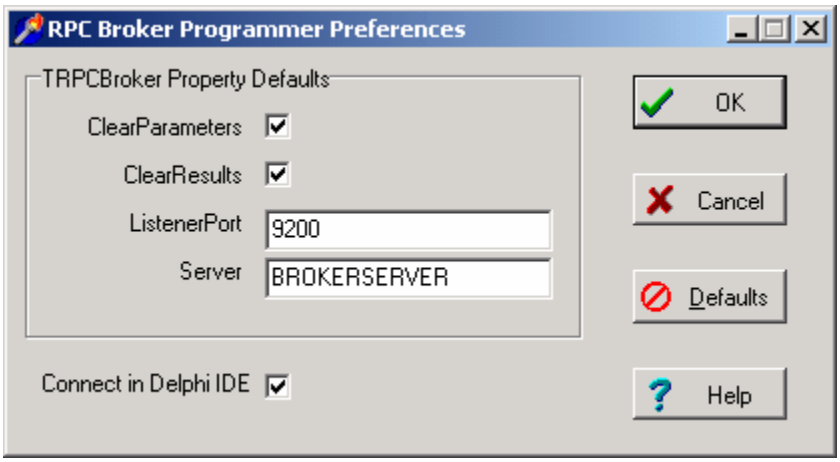

Figure 6-1: RPC Broker Programmer Preferences dialogue

You may want to make an entry for BrokerProgPref.EXE in Delphi's Tools Menu, to make it easily accessible from within Delphi.

Setting	Description
ClearParameters	If checked, sets the ClearParameters property of a TRPCBroker component to True when you add one to a form.
ClearResults	If checked, sets the ClearResults property to of a TRPCBroker component to True when you add one to a form.
ListenerPort	Sets the ListenerPort property of a TRPCBroker component to the specified value when you add one to a form.
Server	Sets the Server property of a TRPCBroker component to the specified value when you add one to a form.
Connect in Delphi IDE	Enables or disables the connection to the M server from within the Borland Integrated Development Environment (IDE). Disabling this is useful when you are developing in an environment without a connection to an M server. For example, when editing certain server properties, an attempt is made to connect to the Server (if enabled).

Table 6-1: Programmer preference settings

7. RPC Broker and Delphi

The following topics highlight changes made to or comments about the Broker to accommodate a particular version of Delphi. They are listed in reverse Delphi version order.

Delphi 6.0 Packages

:

Delphi V. 6 Standard Edition *Not* Recommended for BDK Development

Delphi V. 6.0 comes in three flavors: Standard, Professional, and Enterprise. It is recommended that either the Professional or Enterprise version of Delphi 6.0 be used to develop applications using the RPC Broker.

For more information on the different editions of Delphi, please refer to the "Delphi V. 5 Standard Edition Not Recommended for BDK Development" topic below.

XWB_Rxx.BPL File

The installation of Patch XWB*1.1*13 removed the run-time bpl for the Broker package, if it is present in the System32 directory. This run-time package is found in the Delphi6\Projects directory. While on client workstations, the proper location for the run-time bpl is in the System32 directory (so that it is in the system path), however, on developer workstations, this frequently leads to problems when packages are recompiled and the newly generated bpl file is not moved into this directory. To simplify matters on the developer workstation, it is best that the file *not* be put into the System32 directory.

Delphi 5.0 Packages

:

Delphi V. 5 Standard Edition *Not* Recommended for BDK Development

Delphi V. 5.0 comes in three flavors: Standard, Professional, and Enterprise. This version of the BDK requires the Professional or Enterprise Edition. Standard edition is targeted mainly at students, and as such leaves out many features. We do *not* recommend using the Standard edition of Delphi V. 5.0 for RPC Broker development at this time:

1. Delphi V. 5.0 Standard Edition does not ship with the OpenHelp help system, whose purpose is to allow easy integration of 3rd party component help with Delphi's own internal component help.
2. The RPC Broker component has a dependency on a VCL source code unit, "dsgnintf.pas". Delphi V. 5.0 Standard Edition does not ship the "dsgnintf" file, in either .PAS or .DCU form. VCL Source code units are available in Delphi 5 Professional and Enterprise Editions. When installing Delphi V. 5.0 Professional or Enterprise Edition, make sure you leave the VCL Source installation option selected.

XWB_Rxx.BPL File

The installation of Patch XWB*1.1*13 removed the run-time bpl for the Broker package, if it is present in the System32 directory. This run-time package is found in the Delphi5\Projects\Bpl directory. While on client workstations, the proper location for the run-time bpl is in the System32 directory (so that it is in the system path), however, on developer workstations, this frequently leads to problems when packages are recompiled and the newly generated bpl file is not moved into this directory. To simplify matters on the developer workstation, it is best that the file *not* be put into the System32 directory.

Delphi 4.0 Packages

:
Patch XWB*1.1*14 split the Broker package into separate run- and design-time packages. If a package is defined with the VistaBroker.DPK as a Required package, you must delete that required package and add XWB_Dxx.DPK (where xx=30 for Delphi 3.0, =40 for Delphi 4.0, or =50 for Delphi 5.0) as a required package.

XWB_Rxx.BPL File

The installation of Patch XWB*1.1*13 removed the run-time bpl for the Broker package, if it is present in the System32 directory. This run-time package is found in the Delphi4\Bin directory. While on client workstations, the proper location for the run-time bpl is in the System32 directory (so that it is in the system path), however, on developer workstations, this frequently leads to problems when packages are recompiled and the newly generated bpl file is not moved into this directory. To simplify matters on the developer workstation, it is best that the file *not* be put into the System32 directory.

Delphi 3.0 Packages

:
Delphi V. 3.0 enabled applications to be distributed in specially compiled dynamic-link libraries called Delphi Package Libraries (DPLs). DPLs enable code sharing, reduction of executable file size, and conservation of system resources. The use of DPLs is restricted to development in Delphi V. 3.0 or higher.

VistaBroker.DPL

You can compile the TRPCBroker component code into your application, or you can choose instead to link your application to the VistaBroker.DPL dynamic link library.

For more information on how to use DPLs, please refer to the "Borland Delphi 3 User's Guide."

The VistaBroker.DPL file is installed on end-user workstations by the RPC Broker V. 1.1 end-user client workstation installation program, along with the other RPC Broker V. 1.1. client files. It is installed in an appropriate directory, depending on the operating system (e.g., \WINDOWS\SYSTEM) such that the DPL is accessible when an application calls it. Therefore, you do not need to distribute VistaBroker.DPL with your application.

Distributing the Delphi VCL30.DPL

If you choose to access functionality of the TRPCBroker component through the VistaBroker.DPL dynamic link library, an additional requirement is that the VCL30.DPL library (provided with Delphi V. 3.0) be installed on the end-user workstation.

The RPC Broker does not distribute VCL30.DPL to end-user workstations (it only distributes VistaBroker.DPL). You must ensure that VCL30.DPL is also installed on end-user workstations, perhaps through the installation instructions you provide to system managers.

8. RPC Broker Dynamic Link Library (DLL)

DLL Interface

The RPC Broker provides a Dynamic Link Library (DLL) interface, which acts like a "shell" around the Delphi TRPCBroker component. The DLL is contained in the BAPI32.DLL file.

The DLL interface enables client applications, written in any language that supports access to Microsoft Windows DLL functions, to take advantage of all features of the TRPCBroker component. This allows programming environments other than Borland Delphi to make use of the TRPCBroker component. All of the communication to the server is handled by the TRPCBroker component, accessed via the DLL interface.

Exported Functions

The complete list of functions exported in the DLL is provided in the *RPC Broker Developer's Guide* (i.e. BROKER.HLP, online help distributed with the BDK). Functions are provided in the DLL for:

- Creating and destroying RPC Broker components.
- Setting and retrieving RPC Broker component properties.
- Executing RPC Broker component methods.

Header Files Provided

The following header files provide correct declarations for DLL functions:

C	BAPI32.H
C++	BAPI32.HPP
Visual Basic	BAPI32.BAS

Sample DLL Application

The Visual Basic (VB) EGCHO sample application distributed with the Broker Development Kit (see the ../BDK32/Samples/Vb5Egcho directory), demonstrates use of the TRPCBroker DLL from Microsoft Visual Basic.

Return Values from RPCs

Results from an RPC executed on an M server are returned as a text stream. This text stream may or may not have embedded **<CR><LF>** character combinations.

When you call an RPC using the TRPCBroker component for Delphi, the text stream returned from an RPC is automatically parsed and returned in the TRPCBroker component's Results property as follows:

Results stream contains <CR><LF> combinations	**Location/format of results** (assumes RPC's WORD WRAP ON field is True if RPC is Global Array or Word-processing type)
Yes	Results nodes, split based on <CR><LF> delimiter
No	Results[0]

Table 8-1: TRPCBroker component's Results property

When you call an RPC using the DLL interface, the return value is the unprocessed text stream, which may or may not contain **<CR><LF>** combinations. It is up to you to parse out what would have been individual Results nodes in Delphi, based on the presence of any **<CR><LF>** character combinations in the text stream.

COTS Development and the DLL

The Broker DLL serves as the gateway to the REMOTE PROCEDURE file (#8994) for non-Delphi client/server applications. In order to use any RPCs not written specifically by the client application (e.g., CONSULTS FOR A PATIENT, USER SIGN-ON RPCs, or the more generic VA FileMan RPCs), you must call the RPC Broker DLL with input parameters defined and results accepted in the formats required by the RPC being called.

Therefore, to use the Broker DLL interface you must determine the following information for each RPC you plan to use:

- How does the RPC expect input parameters, if any, to be passed to it?
- Will you be able to create any input arrays expected by the RPC in the same format expected by the RPC?
- What will the results data stream returned by the RPC look like?

Glossary

CLIENT	A single term used interchangeably to refer to the user, the workstation, and the portion of the program that runs on the workstation. In an object-oriented environment, a client is a member of a group that uses the services of an unrelated group. If the client is on a local area network (LAN), it can share resources with another computer (server).
COMPONENT	An object-oriented term used to describe the building blocks of GUI applications. A software object that contains data and code. A component may or may not be visible. These components interact with other components on a form to create the GUI user application interface.
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol.
DLL	**D**ynamic **L**ink **L**ibrary. A DLL allows executable routines to be stored separately as files with a DLL extension. These routines are only loaded when a program calls for them. DLLs provide several advantages: 1. DLLs help save on computer memory, since memory is only consumed when a DLL is loaded. They also save disk space. With static libraries, your application absorbs all the library code into your application so the size of your application is greater. Other applications using the same library will also carry this code around. With the DLL, you don't carry the code itself, you have a pointer to the common library. All applications using it will then share one image. 2. DLLs ease maintenance tasks. Because the DLL is a separate file, any modifications made to the DLL will not affect the operation of the calling program or any other DLL. 3. DLLs help avoid redundant routines. They provide generic functions that can be utilized by a variety of programs.
GUI	**G**raphical **U**ser Interface. A type of display format that enables users to choose commands, initiate programs, and other options by selecting pictorial representations (icons) via a mouse or a keyboard.
ICON	A picture or symbol that graphically represents an object or a concept.
REMOTE PROCEDURE CALL	A remote procedure call (RPC) is essentially M code that may take optional parameters to do some work and then return either a single value or an array back to the client application.

SERVER	The computer where the data and the Business Rules reside. It makes resources available to client workstations on the network. In VistA, it is an entry in the OPTION file (#19). An automated mail protocol that is activated by sending a message to a server at another location with the "S.server" syntax. A server's activity is specified in the OPTION file (#19) and can be the running of a routine or the placement of data into a file.
USER ACCESS	This term is used to refer to a limited level of access to a computer system that is sufficient for using/operating software, but does not allow programming, modification to data dictionaries, or other operations that require programmer access. Any of VistA 's options can be locked with a security key (e.g., XUPROGMODE, which means that invoking that option requires programmer access). The user's access level determines the degree of computer use and the types of computer programs available. The Systems Manager assigns the user an access level.
USER INTERFACE	The way the software is presented to the user, such as Graphical User Interfaces that display option prompts, help messages, and menu choices. A standard user interface can be achieved by using Borland's Delphi Graphical User Interface to display the various menu option choices, commands, etc.
WINDOW	An object on the screen (dialogue) that presents information such as a document or message.

For a comprehensive list of commonly used infrastructure- and security-related terms and definitions, please visit the ISS Glossary Web page at the following Web address:

http://vista.med.va.gov/iss/glossary.asp

For a comprehensive list of acronyms, please visit the ISS Acronyms Web site at the following Web address:

http://vista/med/va/gov/iss/acronyms/index.asp

Index

$

$$BROKER^XWBLIB, 4-4
$$RTRNFMT^XWBLIB, 4-5

A

About this Version of the BDK, 1-1
Acknowledgements, x
Acronyms (ISS)
  Home Page Web Address, Glossary, 2
AllowShared Property, 2-5
APIs
  $$BROKER^XWBLIB, 4-4
  $$RTRNFMT^XWBLIB, 4-5
Application.Run Method, 4-2
Assumptions About the Reader, xv

B

Backward Compatibility Issues, 1-2
BAPI32.BAS File, 8-1
BAPI32.DLL File, 8-1
BAPI32.H File, 8-1
BAPI32.HPP File, 8-1
BROKER.HLP, xvi
BrokerExample, 3-7
BROKEREXAMPLE.EXE, 3-7
BrokerProgPref.EXE, 6-1
Bypassing Security for Development, 3-7

C

C Language, 8-1
C++ Language, 8-1
Call Method, 2-2, 3-6
Calls
  Discrete, 3-5
  Silent, 3-5
ClearParameters Property, 2-2, 6-1
ClearResults Property, 2-2, 6-1
Commonly Used Terms, xiii
Compatibility Issues, 1-2
Components
  RPC Broker Components for Delphi, 2-1
  TCCOWRPCBroker, 2-4, 2-5
  TRPCBroker, 2-1
  TSharedBroker, 2-5
  TSharedRPCBroker, 2-5
  TXWBRichEdit, 2-6
Connect in Delphi IDE, 6-1
Connect To, 4-1
Connected Property, 2-2
Connection
  Testing Your RPC Broker Connection, 5-2
Contents, v
Context-sensitive Help, xvi
COTS Development and the DLL, 8-2
Create Your Own RPCs, 3-1
CreateContext Method, 2-3, 3-7

D

Debugging, 5-1
  Error Trapping, 5-1
  How to Debug Your Client Application, 5-1
  Identifying
    Handler Process on the Server, 5-1
    Listener Process on the Server, 5-1
  Testing Your RPC Broker Connection, 5-2
DebugMode Property, 5-1
DECRYP^XUSRB1, 4-4
Decrypt Method, 4-4
Decryption Functions, 4-4
Delphi
  3.0 Packages, 7-2
  4.0 Packages, 7-2
  5.0 Packages, 7-1
  6.0 Packages, 7-1
Delphi and RPC Broker, 7-1
Delphi Components
  RPC Broker, 2-1
Developer Utilities, 6-1
Developer's Guide
  Online, xvi
Diagnostic Program, 5-2
Discrete Calls, 3-5
Distributing the Delphi VCL30.DPL, 7-3
DLL
  COTS Development and the DLL, 8-2
  Exported Functions, 8-1
  Header Files, 8-1
  Interface, 8-1
  Sample DLL Application, 8-1

Documentation
Revisions, iii
DPL, 7-2
Dynamic Link Library (DLL), 8-1

E

EBrokerError, 5-1
EN^XWBTCP, 5-1
ENCRYP^XUSRB1, 4-4
Encrypt Method, 4-4
Encryption Functions, 4-4
Entry in the Remote Procedure File, 3-5
Error Message Handling, 5-1
EVS Anonymous Directories, xvii
Execute an RPC from a Client Application, How to, 3-5
Exported
DLL Functions, 8-1

F

Fields
NAME (#.01), 3-5
RETURN VALUE TYPE (#.04), 3-5
ROUTINE (#.03), 3-5
TAG (#.02), 3-5
WORD WRAP ON (#.08), 3-2, 3-5, 4-5
Figures and Tables, viii
Files
BAPI32.BAS, 8-1
BAPI32.DLL, 8-1
BAPI32.H, 8-1
BAPI32.HPP, 8-1
Header Files, 8-1
NEW PERSON (#200), 3-3
OPTION (#19), 3-7
REMOTE PROCEDURE (#8994), 3-1, 3-5, 8-2
SECURITY KEY (#19.1), 3-7
VistaBroker.DPL, 7-2, 7-3
XWB_Rxx.BPL, 7-1, 7-2
First Input Parameter for RPCs (Required), 3-2
Functions
Decryption, 4-4
Encryption, 4-4
Exported with DLL, 8-1
Piece, 4-3
Translate, 4-4

G

GetServerInfo Method, 2-2, 2-3, 4-1
Glossary, 1
Glossary (ISS)
Home Page Web Address, Glossary, 2

H

HASH, 4-4
Header Files, 8-1
Help
At Prompts, xiv
Context-sensitive, xvi
Online, xiv
History
Revisions, iii
Home Pages
Adobe Acrobat Quick Guide Web Address, xvi
Adobe Web Address, xvi
Health Systems Design and Development Web Address, xv
ISS Acronyms Home Page Web Address, Glossary, 2
ISS Glossary Home Page Web Address, Glossary, 2
RPC Broker Home Page Web Address, xvi
VistA Documentation Library (VDL) Home Page Web Address, xvi
How to
Connect to an M Server, 2-3
Debug Your Client Application, 5-1
Execute an RPC from a Client Application, 3-5
Obtain Technical Information Online, xiv
Register an RPC, 3-6
Use this Manual, xii

I

Identifying
Handler Process on the Server, 5-1
Listener Process on the Server, 5-1
Input Parameter Types for RPCs (Optional), 3-4
Interface
DLL, 8-1
Introduction, 1-1
ISS Acronyms
Home Page Web Address, Glossary, 2
ISS Glossary
Home Page Web Address, Glossary, 2

Issues
Backward Compatibility, 1-2

L

LAN, 1
List PType, 3-4
ListenerPort Property, 2-2, 4-1, 6-1
Literal PType, 3-4
lstCall Method, 2-3, 3-6

M

M Emulation Functions, 4-3
M Entry Points for RPC Examples, 3-4
Message Handling, Errors, 5-1
Methods
Application.Run, 4-2
Call, 2-2, 3-6
CreateContext, 2-3, 3-7
Decrypt, 4-4
Encrypt, 4-4
GetServerInfo, 2-2, 4-1
lstCall, 2-3, 3-6
SplashClose, 4-2
SplashOpen, 4-2
strCall, 2-3, 3-6
TRPCBroker Component, 2-1
MFUNSTR.PAS, 4-3
Microsoft Windows Registry, 2-4, 4-1

N

NAME Field(#.01), 3-5
NEW PERSON File (#200), 3-3

O

Obtaining
Data Dictionary Listings, xiv
Technical Information Online, How to, xiv
Online
Documentation, xiv
Online Code Samples (RPCs), 3-7
OPTION File (#19), 3-7
Orientation, xii
Other RPC Broker APIs, 4-1

P

Packages
Delphi
3.0 Packages, 7-2
4.0 Packages, 7-2
5.0 Packages, 7-1
6.0 Packages, 7-1
Param Property, 2-2, 3-4, 3-6
Parameters
TimeOut, 4-2
Patches
Revisions, iv
Piece Function, 4-3
Programmer Settings, 6-1
Programs
BROKEREXAMPLE.EXE, 3-7
BrokerProgPref.EXE, 6-1
Properties
AllowShared, 2-5
ClearParameters, 2-2, 6-1
ClearResults, 2-2, 6-1
Connected, 2-2
DebugMode, 5-1
ListenerPort, 2-2, 4-1, 6-1
Param, 2-2, 3-4, 3-6
RemoteProcedure, 2-2, 3-6
Results, 2-2, 3-6
Server, 2-2, 4-1, 6-1
TRPCBroker Component, 2-1
PTypes
List, 3-4
Literal, 3-4
Reference, 3-4

R

Reader, Assumptions About the, xv
Reference Materials, xv
Reference PType, 3-4
Registering RPCs, 3-6
Registry, 2-4, 4-1
Relationship Between an M Entry Point and an RPC, 3-1
Remote Procedure Calls (RPCs), 3-1
REMOTE PROCEDURE File (#8994), 3-1, 3-5, 8-2
RemoteProcedure Property, 2-2, 3-6
Results Property, 2-2, 3-6
RETURN VALUE TYPE Field(#.04), 3-5
Return Value Types for RPCs, 3-2
Return Values from RPCs, 8-2
Revision History, iii
Documentation, iii
Patches, iv

ROUTINE Field (#.03), 3-5
RPC Broker
  Components for Delphi, 2-1
  Home Page Web Address, xvi
RPC Broker and Delphi, 7-1
RPCs, 3-1
  Bypassing Security, 3-7
  Create Your Own RPCs, 3-1
  Error Trapping, 5-1
  Executing, 3-5
  First Input Parameter (Required), 3-2
  Input Parameter Types (Optional), 3-4
  M Entry Point Examples, 3-4
  Online Code Samples, 3-7
  Registering, 3-6
  Relationship Between an M Entry Point and an RPC, 3-1
  Return Value Types, 3-2
  RPC Entry in the REMOTE PROCEDURE File, 3-5
  Security, 3-6
  What is a Remote Procedure Call?, 3-1
  Writing M Entry Points for RPCs, 3-2
  XWB GET VARIABLE VALUE, 4-3
RPCTEST.EXE, 5-2

S

Sample DLL Application, 8-1
Security
  Bypassing Security for Development, 3-7
  How to Register an RPC, 3-6
SECURITY KEY File (#19.1), 3-7
Security Keys
  XUPROGMODE, 3-7
Server Property, 2-2, 4-1, 6-1
Silent Calls, 3-5
Single Signon/User Context (SSO/UC), 2-5
Splash Screen, 4-2
SplashClose Method, 4-2
SplashOpen Method, 4-2
SplVista.PAS Unit, 4-2
SSO/UC, 2-5
Standard Edition, 7-1
strCall Method, 2-3, 3-6
Syntax of GetServerInfo Function, 4-1

T

Table of Contents, v
Tables and Figures, viii
TAG Field (#.02), 3-5
TCCOWRPCBroker Component, 2-4, 2-5
Testing Your RPC Broker Connection, 5-2
TimeOut Parameter, 4-2
Translate Function, 4-4
Trapping RPC Errors, 5-1
Troubleshooting, 5-1
  Connections, 5-1
  Error Trapping, 5-1
  How to Debug Your Client Application, 5-1
  Identifying
    Handler Process on the Server, 5-1
    Listener Process on the Server, 5-1
  Testing Your RPC Broker Connection, 5-2
TRPCBroker Component, 2-1–2-6
  Call Method, 2-2, 3-6
  Connecting to an M Server, 2-3
  CreateContext Method, 2-3, 3-7
  Key Properties, 2-2
  lstCall Method, 2-3
  Methods, 2-2
  Properties and Methods, 2-1
  strCall Method, 2-3
TSharedBroker Component, 2-5
TSharedRPCBroker Component, 2-5
TXWBRichEdit Component, 2-6

U

Units
  SplVista.PAS, 4-2
URLs
  Adobe Acrobat Quick Guide Web Address, xvi
  Adobe Home Page Web Address, xvi
  Health Systems Design and Development Home Page Web Address, xv
  RPC Broker Home Page Web Address, xvi
Using
  Adobe Acrobat Reader, xvi
Utilities, 6-1

V

VBEGCHO, 8-1
Version
  About this Version of the BDK, 1-1
VistA Documentation Library (VDL)
  Home Page Web Address, xvi
VistA Splash Screen, 4-2
VistaBroker.DPL File, 7-2, 7-3

Visual Basic Language, 8-1

W

Web Pages
  Adobe Acrobat Quick Guide Web Address, xvi
  Adobe Home Page Web Address, xvi
  Health Systems Design and Development Home Page Web Address, xv
  ISS Acronyms Home Page Web Address, Glossary, 2
  ISS Glossary Home Page Web Address, Glossary, 2
  RPC Broker Home Page Web Address, xvi
  VistA Documentation Library (VDL) Home Page Web Address, xvi
What is a Remote Procedure Call?, 3-1
What Makes a Good Remote Procedure Call?, 3-5
Windows Registry, 2-4, 4-1
WORD WRAP ON Field (#.08), 3-2, 3-5, 4-5
Writing M Entry Points for RPCs, 3-2

X

XUPROGMODE Security Key, 3-7
XWB GET VARIABLE VALUE RPC, 4-3
XWB_Rxx.BPL File, 7-1, 7-2
XWBLIB
  $$BROKER^XWBLIB, 4-4
  $$RTRNFMT^XWBLIB, 4-5